2016 23rd International Conference on Pattern Recognition (ICPR)
Cancún Center, Cancún, México, December 4-8, 2016

Robust Online 3D Reconstruction Combining a Depth Sensor and Sparse Feature Points

Erik Bylow*, Carl Olsson*, Fredrik Kahl†*
*Center for Mathematical Sciences, Lund University, Lund, Sweden
† Department of Signals and Systems, Chalmers University of Technology
Email: erikb@maths.lth.se, calle@maths.lth.se fredrik.kahl@chalmers.se

***Abstract*—Online 3D reconstruction has been an active research area for a long time. Since the release of the Microsoft Kinect Camera and publication of KinectFusion [11] attention has been drawn how to acquire dense models in real-time. In this paper we present a method to make online 3D reconstruction which increases robustness for scenes with little structure information and little texture information. It is shown empirically that our proposed method also increases robustness when the distance between the camera positions becomes larger than what is commonly assumed. Quantitative and qualitative results suggest that this approach can handle situations where other well-known methods fail. This is important in, for example, robotics applications like when the camera position and the 3D model must be created online in real-time.**[1]

I. Introduction

The ability to create 3D models from image streams has been an active research area for decades. This is known as Structure from Motion, (SfM), in the computer vision community and as Simultaneous Localization and Mapping, (SLAM) in the robotic community. The typical pipeline in SfM is to find a number of key points in a set of images. Thereafter one tries to optimize the position of the 3D points and the configuration of the cameras. Often Structure from Motion only gives a sparse point cloud. With stereo methods it is possible to create a depth map and a dense 3D model instead of a sparse point cloud.

However, since 2010, depth sensors like the Microsoft Kinect and the Asus Pro Live Sensor has grown more and more popular. The ability to create a depth map for each frame greatly simplifies the problem of creating dense 3D models. Consequently, a lot of research have been done on how to generate dense and accurate models using the information from these sensors.

The applications for these techniques are many. For example one can create a dense 3D model of a room to make measurements. Potentially one would also be able to refurbish the model with different furniture and other items. Robot navigation is another application where both the pose of the camera and the model itself are useful. The robot can use the current position of the camera for localization and the model for path planning.

[1]This work has been funded by the Swedish Research Council (grant no. 2012-4213), the Crafoord Foundation and Scientific Research Council, project no. 2012-4215.

The most prominent paper regarding 3D reconstruction with depth sensors is probably KinectFusion [11]. The authors showed that it is possible to create dense models of medium sized rooms in real-time using GPU computations. To achieve this they use a Truncated Signed Distance Function, (TSDF), to represent the model as in [7]. To find the camera pose they perform ICP [3] on a global point cloud obtained from the TSDF and the new point cloud obtained from the new depth image.

Since then, a lot of papers have been published about creating 3D models using depth sensors. In [13] the camera pose is found by maximizing photo consistency between two consecutive images. This was later improved in [8], however, no 3D model is created. Other methods build on KinectFusion such as [17] which is capable of creating larger scale reconstructions using a rolling volume. Also [5] and [6] are an extension of KinectFusion which improve the tracking compared to KinectFusion. The difference between the latter two and KinectFusion is that no ICP is performed. Instead the distance and texture information in the TSDF is used to directly optimize the camera position. Evaluation on benchmarks [16] shows that this improves the pose estimation significantly. In [5] geometry alone was used, while [6] also takes color information into account as well, which improves accuracy and robustness for planar scenes.

In [14], which is [8] and [15] put together, they obtain a dense 3D model using the tracking from [8]. The evaluation on [16] shows impressive results, but it should be noted that it is really an offline method. The 3D model is not created until all images are captured and the camera pose optimized by using loop-closure and other techniques. Even though the tracking itself might be real-time and the creation of the 3D model equally fast, it is still an offline method. Offline methods have the advantage that they acquire all images before the model is created. This makes it easier to reduce drift of the estimated camera movement using loop-closure and bundle-adjustment for example. In contrast, KinectFusion [11], [5] and [6] for example, solve the online problem. That is, the model is created as new images are obtained which is significantly harder. This is because to reduce drift over larger sequences one must be able to detect it and correct the model and estimated pose as images are captured. This is still an open problem and to our knowledge, [18] is so far the only paper with an approach to handle loop-closure in an online

978-1-5090-4846-5/16/$31.00 ©2016 IEEE

method.

The papers mentioned above perform tracking either by using some form of photo consistency or by using the 3D model itself. Some use only geometry [11], [5], others use only color [13] and some use both color and texture [6], [8] and [18]. The advantage of using both color and geometry is that it can increase accuracy and robustness. For example in a planar scene with plenty of texture, the camera pose can still be estimated if color information is used, but not if only geometry is used. Conversely, in a scene with plenty of structure but hardly any texture, the tracking must be geometry based to work.

In this paper we go a step further and extend [6] by invoking additional information in the form of sparse feature points to improve robustness even more. The approach is to find correspondences between the new image and recently taken images to create an error which is independent of the 3D model. The advantage with this is that in some scenes the 3D model does not provide enough information to recover the pose, but typically you can find feature points under many circumstances. Then this new error term helps to find the correct camera position.

We show experimentally that we can handle planar scenes with little texture where [6] fails. Moreover, it is common to assume that the distance between two consecutive images is small. In this paper we will see that we are less reliant to this assumption, making the algorithm more robust.

II. Creating the 3D model

In this part we describe briefly our approach for creating the 3D model, assuming that the global rotation and translation of the camera is known. We rely on the method proposed by [7]. The main idea is to represent the model implicitly by the zero-level in a TSDF. To represent the TSDF one can either use a uniform grid such that all voxels are equidistant to each other, or one can use an octree like representation as in [6]. The former is easy to implement but requires a lot of memory, while the latter is more memory efficient and allows for larger reconstructions. In this paper we choose a uniform grid for simplicity.

Each voxel V contains information about the distance to the surface, D, a weight of the distance measurement, W, color information for texture, (R, G, B), and a weight of the color measurements, W_c.

The voxels can be thought of as lying in a 3-dimensional matrix with indices (i, j, k). Hence, we denote a voxel V at position (i, j, k) as V_{ijk}. The data in each voxel will also be denoted with subscript, for example with D_{ijk} we mean the distance at voxel (i, j, k).

A. Estimating the TSDF

Given a new depth image I_d and a color image I_c, we must integrate this information into our voxel grid in order to update each voxel about the distance between the voxel and the surface.

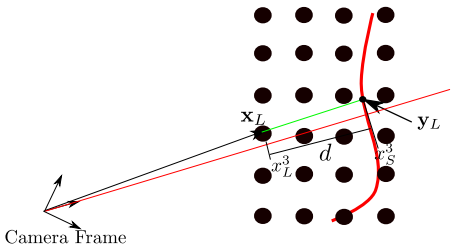


Fig. 1: By computing the difference between $\mathbf{x}_L$ and $\mathbf{y}_L$ along the optical axis we get an approximation of the projected point-to-point distance.

For each voxel V_{ijk} we can compute the 3D-coordinates in the global frame since the voxels are fixed in space. With the known rotation R and translation $\mathbf{t}$ of the camera we can compute the local 3D coordinates for the voxel in the camera's frame of reference

$$\mathbf{x}_L = (x_L, y_L, z_L) = R^T \mathbf{x}_G - R^T \mathbf{t}. \tag{1}$$

Now the voxel can be projected onto the image plane by

$$\begin{aligned} p_x &= \frac{f_x x_L}{z_L} + c_x \\ p_y &= \frac{f_y y_L}{z_L} + c_y, \end{aligned} \tag{2}$$

where f_x, f_y, c_x and c_y are intrinsic camera parameters. Using these pixel coordinates and the depth image I_d, it is possible to read the depth for the surface point by

$$z = I_d(p_x, p_y). \tag{3}$$

We can now estimate the distance as the difference

$$d = z_L - z, \tag{4}$$

which is the distance between the points along the optical axis. This metric is known as the projective point-to-point distance [4], and the method is illustrated in Figure 1.

This simple metric can sometimes give bad estimates. To handle this we truncate the measured distance at a threshold δ by

$$d^{trunc} = \max(-\delta, \min(d, \delta)) \tag{5}$$

and apply a weight $w(d)$, for the measurement as in [5].

With this we get an estimated weighted and truncated distance

$$w(d) d^{trunc}. \tag{6}$$

By doing this for all voxels we get a measurement for each voxel. However, we get information about the distances from all acquired depth images and therefore we need to integrate them into the voxel grid. In [7], this is handled by computing the weighted average of all measurements for each voxel:

$$D = \frac{\sum_{k=1}^{N} w(d_k) d_k^{trunc}}{\sum_{k=1}^{N} w(d_k)}. \tag{7}$$

This makes it easy to integrate a new depth image into the TSDF since we can compute the weighted running average for each voxel.

Similarly, we can compute the color for each voxel. This is done by extracting the RGB-vector from the corresponding color image

$$(r, g, b) = I_c(p_x, p_y). \tag{8}$$

As a weight for the color measurement we take

$$w_c = w(d)\cos(\theta), \tag{9}$$

where θ is the angle between the optical axis and the surface point. Similarly as in (7) we can compute the weighted average of the color for each voxel. This is how we integrate a depth and color image into our TSDF, assuming the global rotation and translation of the camera is known.

III. Estimating the Camera Pose

In this part we describe in detail how the pose of the camera is estimated. Our paper builds on [6] and we give a brief description of the entire pipeline of [6] to simplify for the reader.

The main idea in [6] is to directly use the distance information in the TSDF together with the color information in order to recover the camera position.

Assume that after N images we have a representation of the 3D model through the TSDF. Now we get a new image pair (I_d^{N+1}, I_c^{N+1}). With the depth image I_d^{N+1} we can reconstruct all 3D points in the camera's frame of reference by calculating for each pixel (i, j)

$$\mathbf{x}_L = \begin{pmatrix} \frac{(i-c_x)z}{f_x} \\ \frac{(j-c_y)z}{f_y} \\ z \end{pmatrix}. \tag{10}$$

With a guess of the global rotation R and translation $\mathbf{t}$ of the camera, we can obtain a guess of the global coordinates of $\mathbf{x}_L$ by

$$\mathbf{x}_G = R\mathbf{x}_L + \mathbf{t}. \tag{11}$$

Thus we can reconstruct the point cloud into the TSDF as shown in Figure 2. Most likely the guess of the camera configuration is incorrect so the points will not be reconstructed on the surface. Moreover, we can extract color information in the model at $\mathbf{x}_G$. This color information can be compared to the corresponding rgb-vector in $I_c(i, j)$.

Using this we get a geometric error

$$g(R\mathbf{x}_{ij} + \mathbf{t}) \tag{12}$$

where $\mathbf{x}_{ij}$ is the local 3D point at pixel (i, j) and g is a function that performs trilinear interpolation of the distance value in the TSDF at $R\mathbf{x}_{ij} + \mathbf{t}$, as depicted in Figure 2. We also get the color error

$$h(R\mathbf{x}_{ij} + \mathbf{t}) = \|C(R\mathbf{x}_{ij} + \mathbf{t}) - I_c(i, j)\|, \tag{13}$$

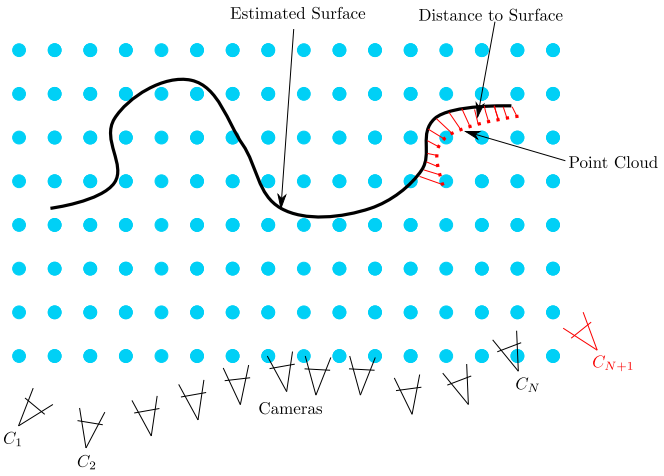


Fig. 2: Given an estimated model we can reconstruct the point cloud from the new depth image into the TSDF, using a guess of the rotation and translation of the camera.

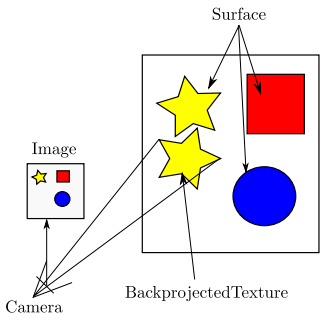


Fig. 3: The reconstructed texture should ideally be aligned to the texture on the model.

where $C(R\mathbf{x}_{ij}+\mathbf{t})$ is the interpolated color vector in the model at $R\mathbf{x}_{ij}+\mathbf{t}$ and $I_c(i, j)$ is the color vector at pixel (i, j). The error is shown in Figure 3. Using this we can minimize

$$E(R, \mathbf{t}) = \sum_{ij} g(R\mathbf{x}_{ij} + \mathbf{t})^2 + \alpha h(R\mathbf{x}_{ij} + \mathbf{t})^2 \tag{14}$$

to find the correct pose.

The combination of distance and texture information significantly improves the robustness of the tracking compared to just using geometry, which was shown in [6]. However, one can easily imagine scenes like corridors or walls where there is little and diffuse texture. In those cases, [6] is unlikely to work since it requires sharp texture on the model.

Therefore, in this paper we add one additional error term, which shall improve robustness for these situations. The idea is to find corresponding key points between the image pair (I_c^{N+1}, I_c^N). This is illustrated in Figure 4.

The advantage of invoking these sparse feature points is that they are not dependent on the model and often feature points are easy to find. Under circumstances where there is not enough information in the model to recover the pose, the sparse feature points can still provide enough constraints to recover the rotation and translation. Another advantage is that

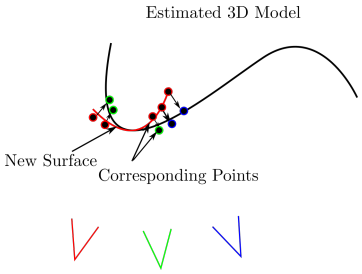


Fig. 4: Finding corresponding points between the newly captured image and previous images with known camera position we can compute an error to minimize.

there exists robust algorithms to find feature points and reject outliers in the matching of the key points in different images.

There exist several methods to find these feature points in an image, for example SIFT [9] and SURF [2]. We choose to work with SURF since it is faster than SIFT and to reject outliers we use RANSAC, which we have found empirically to work reliably.

It is also beneficial to use these feature points when finding R and $\mathbf{t}$ because the distance between the camera positions does not matter when only using corresponding point pairs. In contrast, it is often assumed that the camera motion is small in works like KinectFusion [11], [12] and [6]. By using these feature points, the algorithm should become more robust in situations where the distance between two camera positions are longer than commonly assumed.

Given a new image I_c^{N+1}, we can find feature points in I_c^{N+1} and I_c^N. A 3D point $\mathbf{p}_i^{N+1}$ in I_c^{N+1} then has a corresponding 3D point $\mathbf{q}_i^N$ in I_c^N. Since we know the camera configuration for frame I_c^N, $\mathbf{q}_i^N$ is represented in the global frame of reference. The 3D coordinates $\mathbf{p}_i^{N+1}$ are in the local frame. What we want to minimize is then

$$E_{N+1,N}(R,\mathbf{t}) = \sum_i \|R\mathbf{p}_i^{N+1} + \mathbf{t} - \mathbf{q}_i^N\|^2. \quad (15)$$

Doing the same for the image pair (I_c^{N+1}, I_c^{N-k}), $k = 1 \dots K$, we get a set of corresponding 3D points for each image pair. With these points alone we can define the error function

$$E(R,\mathbf{t}) = \sum_k E_{N+1,N-k}(R,\mathbf{t}). \quad (16)$$

Here we sum over all found corresponding 3D points. $\mathbf{q}_k$ is a 3D point found in any of the images $I_c^N, \dots, I_c^{N-K}$ By combining (14) and (16) we get an error function which combines the information from the TSDF with the error from the corresponding feature points that is independent of the current model. Our new error function thus becomes

$$E(R,\mathbf{t}) = \sum_{ij} g(R\mathbf{x}_{ij} + \mathbf{t})^2 + \alpha h(R\mathbf{x}_{ij} + \mathbf{t})^2 + \mu \sum_{k=0}^{K} \sum_{l=0}^{L_k} \|R\mathbf{x}_l^k + \mathbf{t} - \mathbf{y}_l^k\|^2. \quad (17)$$

Here α and μ are the weighting of the different error terms.

(a) Reconstruction of a corridor using [6]. In this scene one goes out of a room into a corridor and goes straight in the corridor for about 5-6 m. In this reconstruction the corridor is not straight which indicates an incosistent tracking.

(b) Reconstruction of the same scene as in (a) using our proposed method. As can be seen the reconstruction is longer and more straight and looks more like a scene from a corridor.

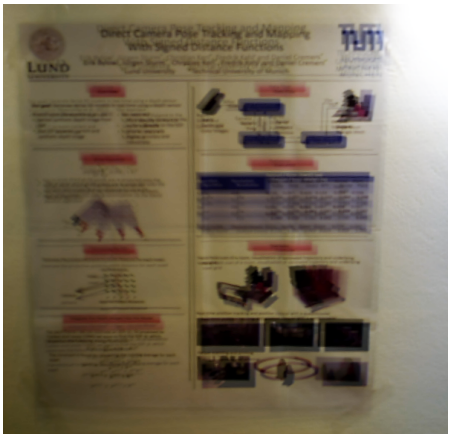
Fig. 5: Comparison between [6] (left) and our proposed method (right).

By using the Lie-Algebra representation [10] of the rotation we can minimize (17) using the Gauss-Newton method.

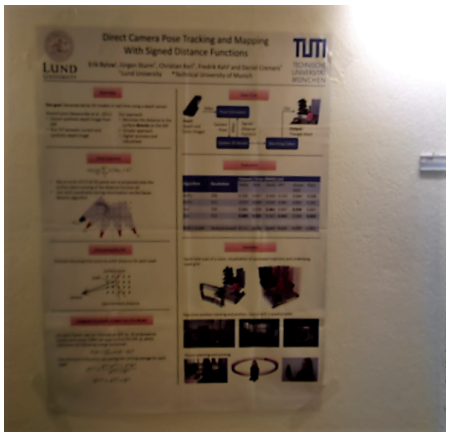
IV. Qualitative Results

To test our proposed algorithm, we start with recording different datasets and test our method on them. In the first experiment we make a recording where the camera is facing the floor. The sequence contains almost no texture and the scene is completely planar. This makes it very challenging, but for example in robot navigation it is important to be able to handle such situations. The sequence recording starts in a room and then it goes into the corridor with a left turn and then it continues in the corridor 5-6 meters. The result for this experiment is shown in Figure 5. As can be seen in Figure 5a, the reconstruction is smaller and looking at the blue squares in the lower left corner the squares are badly reconstructed. This is because when the camera only faces the floor and there is no sharp texture, the algorithm has problem finding the motion between the frames. In contrast, the reconstruction in Figure 5b is larger because it manages to recover the trajectory even when there is no sharp texture. The left turn is also visible and the squares in the blue pattern are better reconstructed. Also one sees in Figure 5b that the floor could lie in a corridor, whereas in Figure 5a the reconstruction is not straight enough.

It could be questioned, why should texture information in the model be used at all? Feature points give constraints for obtaining the pose when the scene is mostly planar. We tested our algorithm on a sequence where we recorded a poster with plenty of texture. Figure 6a shows the result

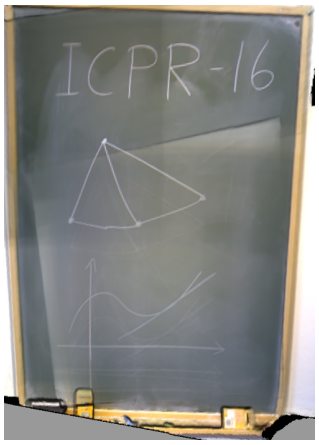


(a) Reconstructed poster using sparse feature points alone. One sees that drift has occured on the letters on the title for example.



(b) Here both information from the model is used together with feature points. The resulting reconstruction is much sharper.

Fig. 6: Comparison between using only feature points and using both feature points and model information. For this sequence just using feature points is not enough.



(a) Without using feature points there is not enough texture to recover the trajectory, one sees clearly that drift is present.

(b) Including feature points as well gives a more accurate result for this sequence.

Fig. 7: In this sequence one can see drift in the left figure, for example the yellow box in the lower left corner is duplicated. In contrast the right reconstruction is sharper, indicating that the pose is better estimated.

using only feature points and Figure 6b shows when one combines both model information and sparse feature points. It is clear that drift occurs when only feature points are used. In contrast, when using information from the model as well, the resulting reconstruction is very sharp which is an indication that the estimated pose of the camera is very accurate. This shows that the information from the 3D model in the TSDF clearly decreases drift. However, as seen in Figure 5a, there is not always enough information in the 3D model to recover the pose. For these experiments the weights $\alpha = 0.4$ and $\mu = 0.75$ were used. In a third experiment we made a recording of a blackboard with not so much texture. As seen in Figure 7a, the texture information is not enough to recover the pose. Instead one sees that the yellow box in the lower left corner is duplicated. In contrast, invoking sparse feature points gives a satisfactory result as shown in Figure 7b. Here the reconstruction is more detailed, which indicates that the found pose is accurately estimated. To get this, μ was set 3.0 and $\alpha = 0.4$.

These experiments show that just using the texture and distance information in the model is not always enough to estimate the pose correctly. However, there are also scenarios where the information in the TSDF is crucial to reduce drift, as in Figure 6b. For applications where robustness is crucial and one might face scenes with little texture and geometry information, one should include as much data as possible.

V. Quantitative Results

In this part we evaluate our algorithm on some benchmarks from [16]. In particular, we will compare our new method to [6] where we simulate faster camera motion. We do this by taking every k:th image in the datasets, starting with $k = 1$. This will test how well we perform when the camera has moved longer between each frame. As can be seen in Table I, our proposed method does not drift away as quickly as [6] does when k is increasing. For example in the dataset Teddy, invoking SURF points makes it much more robust and gives decent results when only using every 4-th frame. In contrast, [6] fails already when using every 2nd frame. This shows that the robustness increases when the camera motion is faster if sparse feature points are included. The weights used in this experiment was $\alpha = 0.4$ and $\mu = 0.75$ and the number of voxels was 512^3. On the benchmarks, the frame rate was about 4-5 Hz using a resolution of 512^3 voxels and the images of size 640×480 pixels. For the recorded sequences the frame rate was about 16-17 Hz with the same number of voxels but with images of resolution 320×240 pixels. The computer used was a Intel i7, 3.4 GHz and 16 GB of RAM.

Dataset \ Step Size k	1	2	3	4	5	6
Teddy [6]	**0.059**	0.473	0.656	1.142	1.098	0.990
Teddy Our	0.060	**0.059**	**0.067**	**0.065**	**0.274**	**0.268**
Desk2 [6]	0.117	0.302	**0.442**	0.820	0.715	0.991
Desk2 Our	**0.058**	**0.083**	0.526	**0.317**	**0.289**	**0.507**
360 [6]	0.131	0.288	0.773	1.419	1.774	1.939
360 Our	**0.102**	**0.010**	**0.206**	**0.581**	**1.493**	**1.254**
Plant [6]	**0.045**	0.069	0.290	0.351	0.448	**0.531**
Plant Our	0.047	**0.046**	**0.215**	**0.348**	**0.111**	0.548
Desk [6]	**0.032**	0.051	0.094	0.244	0.422	0.603
Desk Our	**0.032**	**0.033**	**0.037**	**0.045**	**0.100**	**0.320**

TABLE I: Results on the benchmarks from [16]. We use every k-th image and compute the RMSE (m) to test the robustness for bigger distances between two consecutive frames.

The graphics card used was an NVidia Geforce GTX 770. The methods from [6] were run on the GPU. Our extension with finding corresponding feature points is not run in parallel. The speed can potentially be improved by using parallelization and maybe using GPU implementations of SURF or SIFT. In [5] it was shown that our original method outperformed KinFu [1]. For a more detailed evaluation for other methods, please see [5] and [6] .

VI. Conclusion

In this paper we have shown that including an additional error term using sparse feature points to [6] significantly can increase the robustness in situations where there is little geometric scene structure and texture information. Experiments also suggest that additionally using sparse feature points increases accuracy when the distance between two consecutive images becomes larger. A clear drawback is that the weights α and μ must be set manually. For practical applications, it would be beneficial if the weights could be tuned automatically.

References

[1] KinectFusion Implementation in the Point Cloud Library (PCL). `http://svn.pointclouds.org/pcl/trunk/`.

[2] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. In *In ECCV*, pages 404–417, 2006.

[3] P.J. Besl and N.D. McKay. A method for registration of 3-D shapes. *IEEE Trans. Pattern Anal. Mach. Intell.*, 14 (2):239–256, 1992.

[4] G. Blais and M.D. Levine. Registering multiview range data to create 3D computer objects. *IEEE Trans. Pattern Anal. Mach. Intell.*, 17:820–824, 1993.

[5] E. Bylow, J. Sturm, C. Kerl, F. Kahl, and D. Cremers. Real-time camera tracking and 3d reconstruction using signed distance functions. In *RSS*, 2013.

[6] Erik Bylow, Carl Olsson, and Fredrik Kahl. Robust camera tracking by combining color and depth measurements. In *Pattern Recognition (ICPR), 2014 22nd International Conference on*, pages 4038–4043. IEEE, 2014.

[7] B. Curless and M. Levoy. A volumetric method for building complex models from range images. In *SIGGRAPH*, 1996.

[8] Christian Kerl, Jürgen Sturm, and Daniel Cremers. Dense visual slam for rgb-d cameras. 2013.

[9] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60:91–110, 2004.

[10] Y. Ma, S. Soatto, J. Kosecka, and S. Sastry. *An Invitation to 3D Vision: From Images to Geometric Models*. Springer Verlag, 2003.

[11] R.A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A.J. Davison, P. Kohli, J. Shotton, S. Hodges, and A.W. Fitzgibbon. KinectFusion: Real-time dense surface mapping and tracking. October 2011.

[12] F. Steinbrücker, J. Sturm, and D. Cremers. Real-time visual odometry from dense rgb-d images. In *Workshop on Live Dense Reconstruction with Moving Cameras at ICCV*, 2011.

[13] F. Steinbruecker, J. Sturm, and D. Cremers. Real-time visual odometry from dense rgb-d images. In *Workshop on Live Dense Reconstruction with Moving Cameras at the Intl. Conf. on Computer Vision (ICCV)*, 2011.

[14] F. Steinbruecker, C. Kerl, J. Sturm, and D. Cremers. Large-scale multi-resolution surface reconstruction from rgb-d sequences. In *IEEE International Conference on Computer Vision (ICCV)*, Sydney, Australia, 2013.

[15] F. Steinbruecker, J. Sturm, and D. Cremers. Volumetric 3d mapping in real-time on a cpu. Hongkong, China, 2014.

[16] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers. A benchmark for the evaluation of RGB-D SLAM systems. In *IROS*, 2012.

[17] T. Whelan, H. Johannsson, M. Kaess, J.J. Leonard, and J.B. McDonald. Robust real-time visual odometry for dense RGB-D mapping. In *IEEE Intl. Conf. on Robotics and Automation, ICRA*, Karlsruhe, Germany, May 2013.

[18] Thomas Whelan, Stefan Leutenegger, Renato F Salas-Moreno, Ben Glocker, and Andrew J Davison. Elasticfusion: Dense slam without a pose graph. In *Robotics: Science and Systems (RSS)*, 2015.